Control Flow

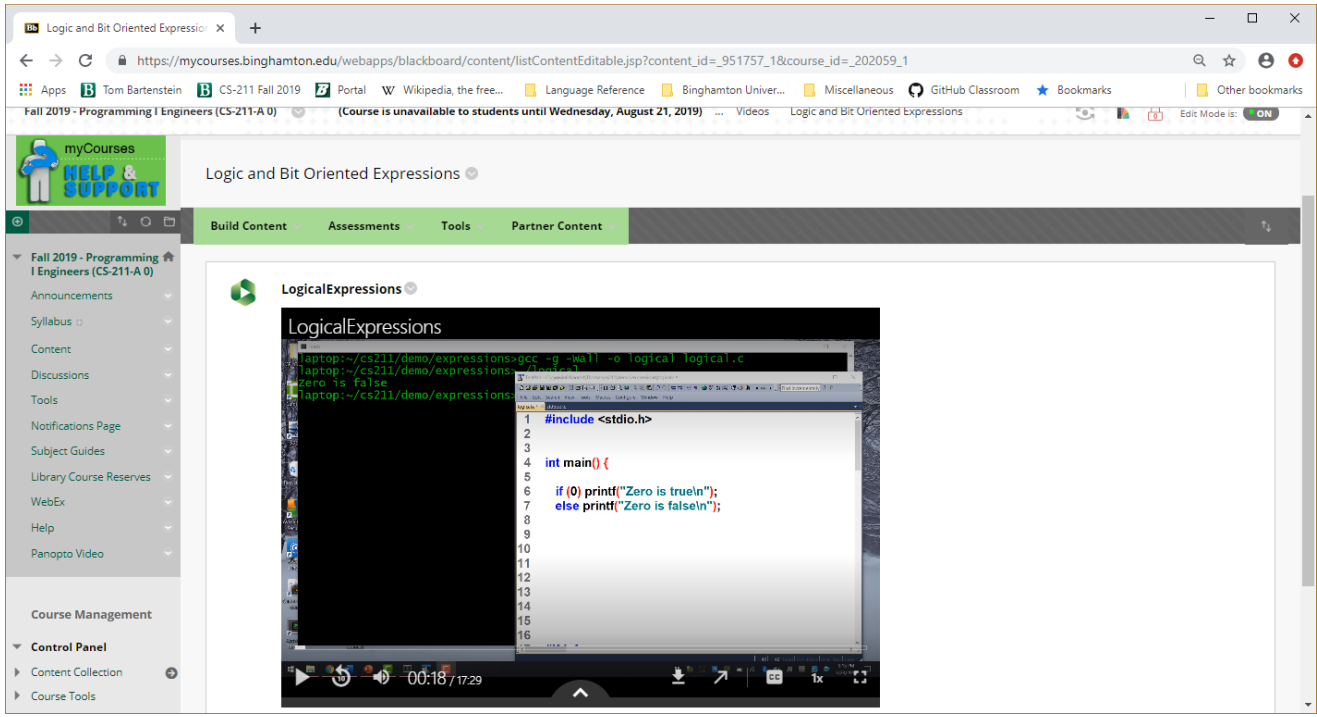
iClicker Attendance

Please click on A if you are here:

A. I am here today.

Video Review: Logical Expressions

- Truth values of numbers
- Conditional Expressions
- Comparison Operators: <. <=. ==, !=. >=. >
  - Comparing Float
- Logical Operators: !, &&, ||
- Ternary Operator: ? :



iClicker Question

- What gets printed?

```
int x=83;
if (x=85) printf("x was 85… ");
else printf("x was not 85… ");
printf("x is now %d\n",x);
```

A. No value – compiler error.
B. x was not 85… x is now 83
C. x was not 85… x is now 85
D. x was 85… x is now 85

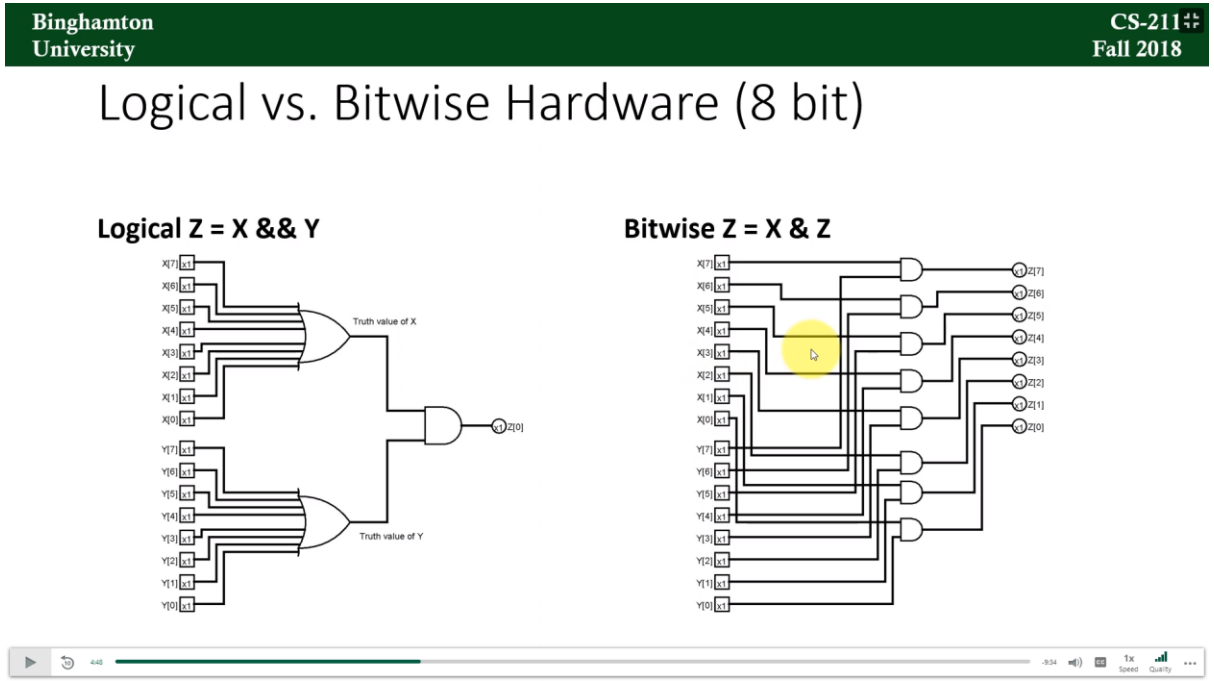
Class Exercise: Condition Evaluation

- What does the following code print when compiled and executed?

```
int a = 13;
int b = ++a;
if (++b < ++a) { printf("Condition 1\n"); }
else {
   if (--b > a--) { printf("Condition 2\n"); }
   else {
      if (++b < a++) { printf("Condition 3\n"); }
      else { printf("Condition 4\n"); }
   }
}
printf("a=%d, b=%d\n",a,b);
```

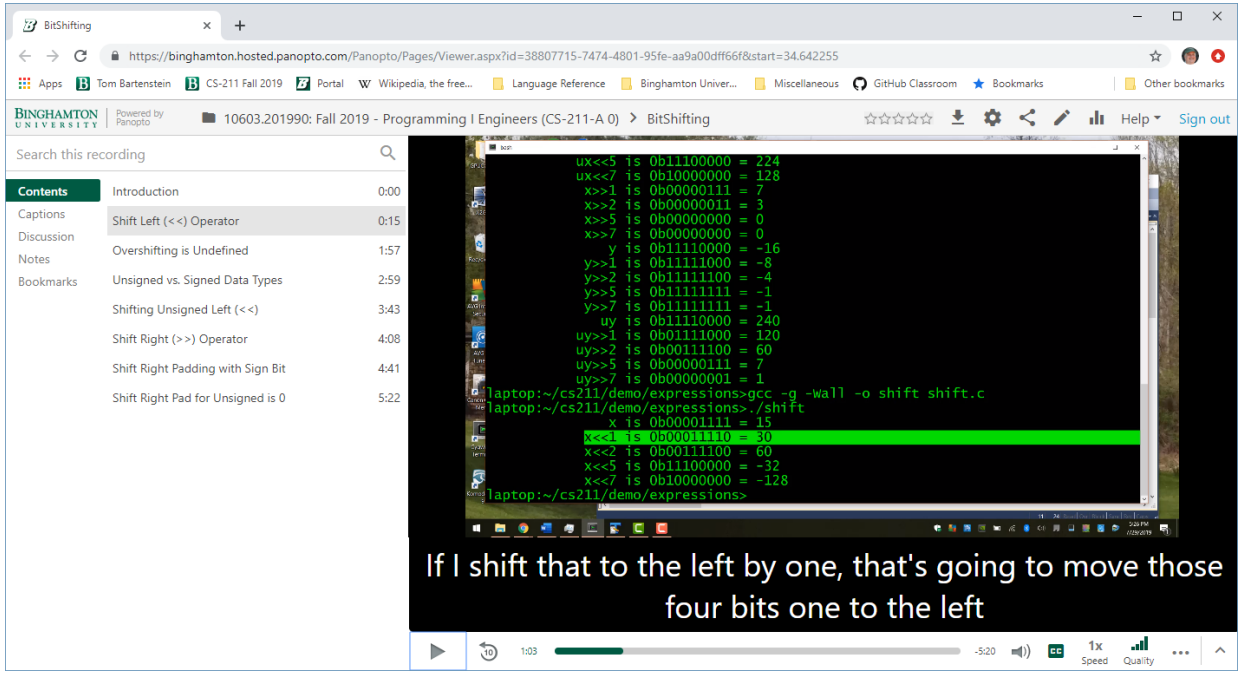
Video Review: Bit Oriented Expressions

- Logical vs. Bitwise Operators / Expressions
- Bitwise Operators: ~, &, |, ^
- Bitwise Semantics



Video Review: Bit Shifting

- Shift Operators: >> and <<
  - Overshift pitfall!
- << left side padding
- >> right side padding
  - Positive numbers
  - Negative numbers
  - Unsigned numbers



(Project LookAhead) Arguments to main

- When the operating system calls the “main” function, it:
- parses the command line, splitting at “white space” (blanks, tabs)
  >./convertTemp 21.3 F C
- Counts the number of blank delimited words: argc=4
- Creates an array of “words” or strings (really an array of arrays)

argv[0]	“./convertTemp”
argv[1]	“21.3”
argv[2]	“F”
argv[3]	“C”

Size=4

(Project LookAhead) Parameters of main

- Typically use argc and argv as the names of the parameters to main

```
int main(int argc, char **argv) { …
```

- argc argument count – the size of the argv array
- argv argument value (or vector) - a list of strings
  - Each string is an array of characters
  - So argv is an array of arrays of characters
  - Can reference each string as argv[i]
  - Can reference individual letters as argv[i][j]
  - For instance, the first letter of the second arg is argv[1][0]

"Bit Twiddling"

- Combination of bitwise operations and shifting
- Enables manipulation of multiple bits
- Often very "clever" (or confusing)
- Examples…

```
x <<=y; // multiply by 2^y
x >>=3; // Divide by 8 (almost)
if ((x^y)<0) // do x and y have opposite signs?
for(c=0;v;v>>=1) c+=v&1; // count true bits in v
```

Difference between shift right and divide

```
char x = -15; // 0b 1111 0001
char y = x/8; // -15/8 = -1.825 = -1 = 0b 1111 1111
char z = x>>3; // 0b 1111 1000 ->
               // 0b 1111 1100 ->
               // 0b 1111 1110 = -2
```

- Integer division always rounds towards zero
  - Rounds down for positive numbers
  - Rounds up for negative numbers
- Shift right always truncates
  - Always rounds down for both positive and negative numbers

Class Exercise: Bit Flags

- You have variables my_speed and your_speed to tell how fast I am and how fast you are.
- You have variables my_press and your_press to tell how much weight you and I can bench press at the gym
- Create a char variable called “flags” that has:
  - A flag bit to indicate whether I am faster or you are faster
  - A flag bit to indicate whether I am stronger or you are stronger
  - A flag bit to indicate whether I am better (both faster and stronger) than you are
  - A flag bit to indicate that the other flag bits have been set on

Using Bit Twiddling for Multiple Flags

```
#define IS_FASTER 0b00000001 // decimal value 1
#define IS_STRONGER 0b00000010 // decimal value 2
#define IS_BETTER 0b00000100 // decimal value 4
#define IS_SET 0b00001000 // decimal value 8

char flags=0; // Init all flags off
if (my_speed > your_speed) flags |= IS_FASTER; // turn on faster flag
if (my_press  > your_press) flags |= IS_STRONGER; // turn on stronger flag
if (flags & IS_FASTER && flags & IS_STRONGER) flags |= IS_BETTER; // turn on better
flags |=IS_SET; // turn on set flag
```

Class Exercise: Bit Flags (Continued)

- If variable “my_ego” is false, turn off the IS_BETTER flag

- If they are true, then print out
  - “I am faster”
  - “I am stronger”
  - “I am better”

Using Bit Twiddling for Multiple Flags

```
#define IS_FASTER 0b00000001
#define IS_STRONGER 0b00000010
#define IS_BETTER 0b00000100
#define IS_SET 0b00001000
…
if (!my_ego) flags &= ~IS_BETTER; // Turn off IS_BETTER flag
if (flags & IS_FASTER) printf("I am faster.\n");
if (flags & IS_STRONGER) printf("I am stronger.\n");
if (flags & IS_BETTER) printf("I am better.\n");
```

Exercise Review: Conv. to Binary & Hex

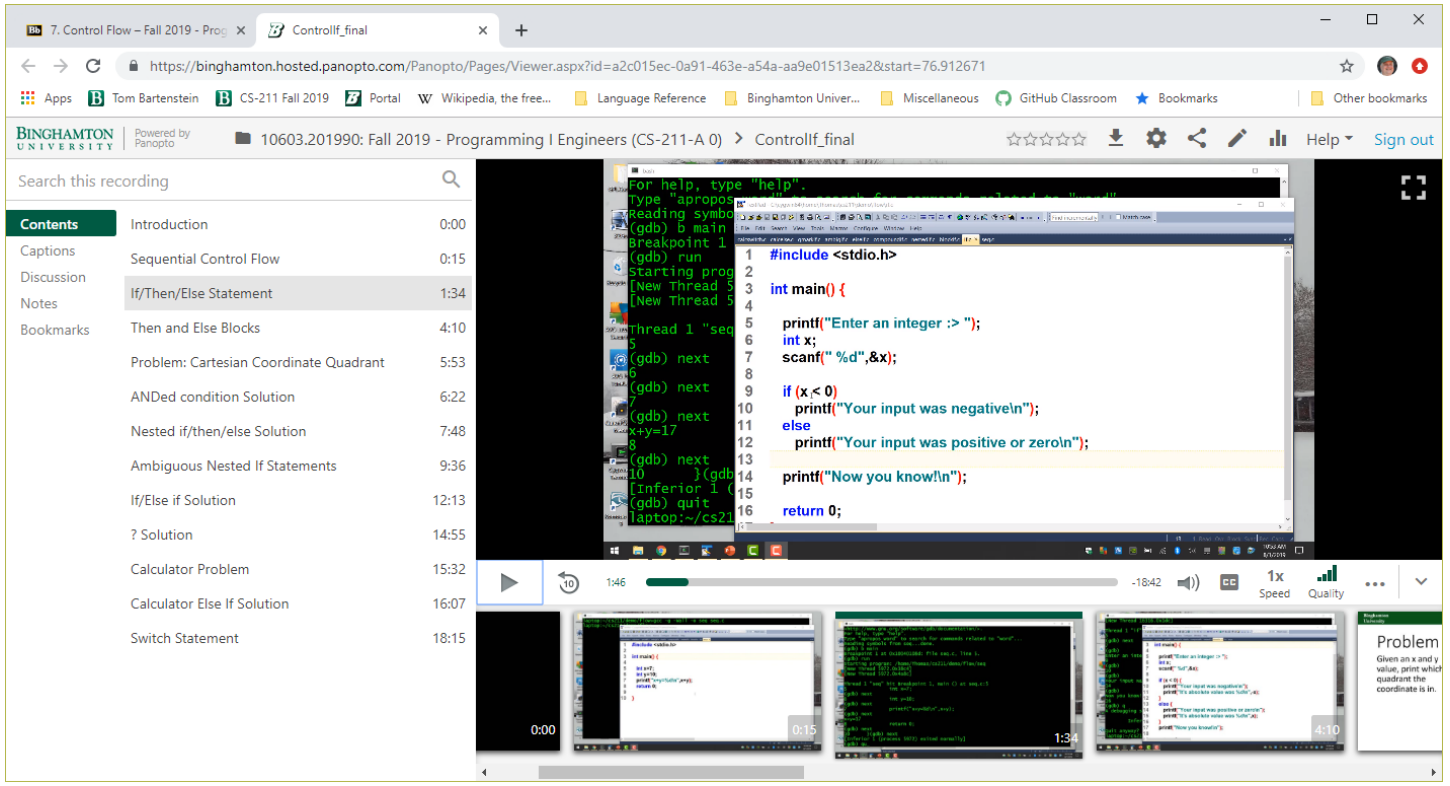
Get an integer number from the user.

Write out the binary representation of that number.

Write out the hexadecimal representation of that number (without using printf %x format).

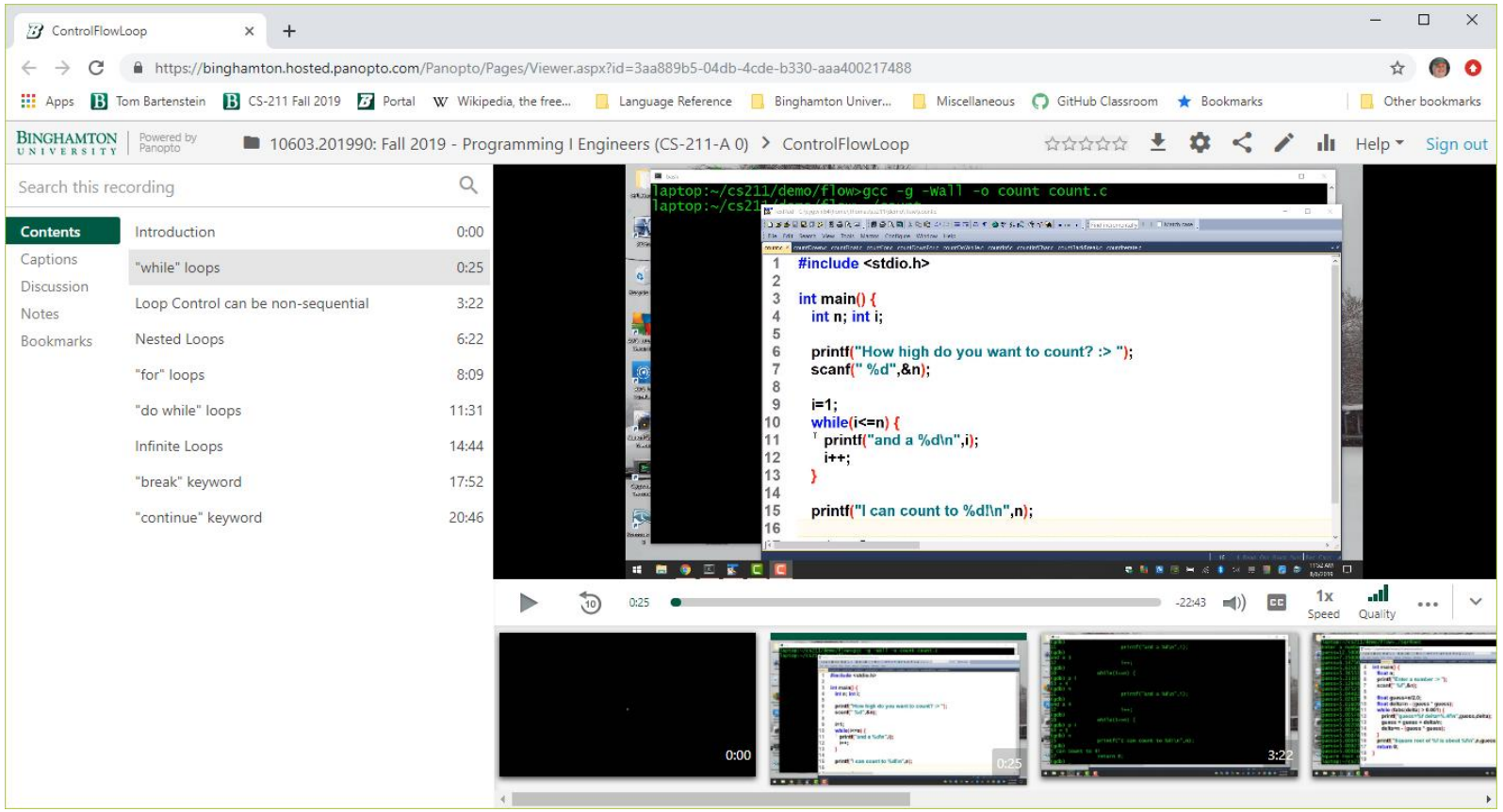
Control Flow: If Statements

- Watch the video, available on myCourses
  - Content
    - Videos
      - 7. Control Flow



Control Flow: Loops

- Watch the video, available on myCourses
  - Content
    - Videos
      - 7. Control Flow



Exercise: HighLow

Write a computer program that selects a random number between 0 and 100. Then prompt the user to enter a guess of that number. As long as the user has not guessed the number, tell her whether her guess is greater than or less than the target number, then ask for another guess.

Exercise Background : Random Numbers

- The standard library defined by <stdlib.h> contains a built-in function called “rand()”, which returns a pseudo-random non-negative integer
- You can get a number between 0 and 100 by doing: `rand()%100`
- The pseudo-random seed is “1” by default, so you get the same random number each time, unless we change the random number seed…

Exercise Background: Random Seed

- To approximate true randomness, use the current time as a random seed.
- The time capabilities are contained in standard library header <sys/times.h>.
- The “times()” built-in function takes an argument that specifies space to use for the result, and returns the current time, in processor clock ticks since last boot.
- The standard library “srand(seed)” function allows you to tell the random number generator to use a new seed (instead of 1)

Exercise Background: Example Random

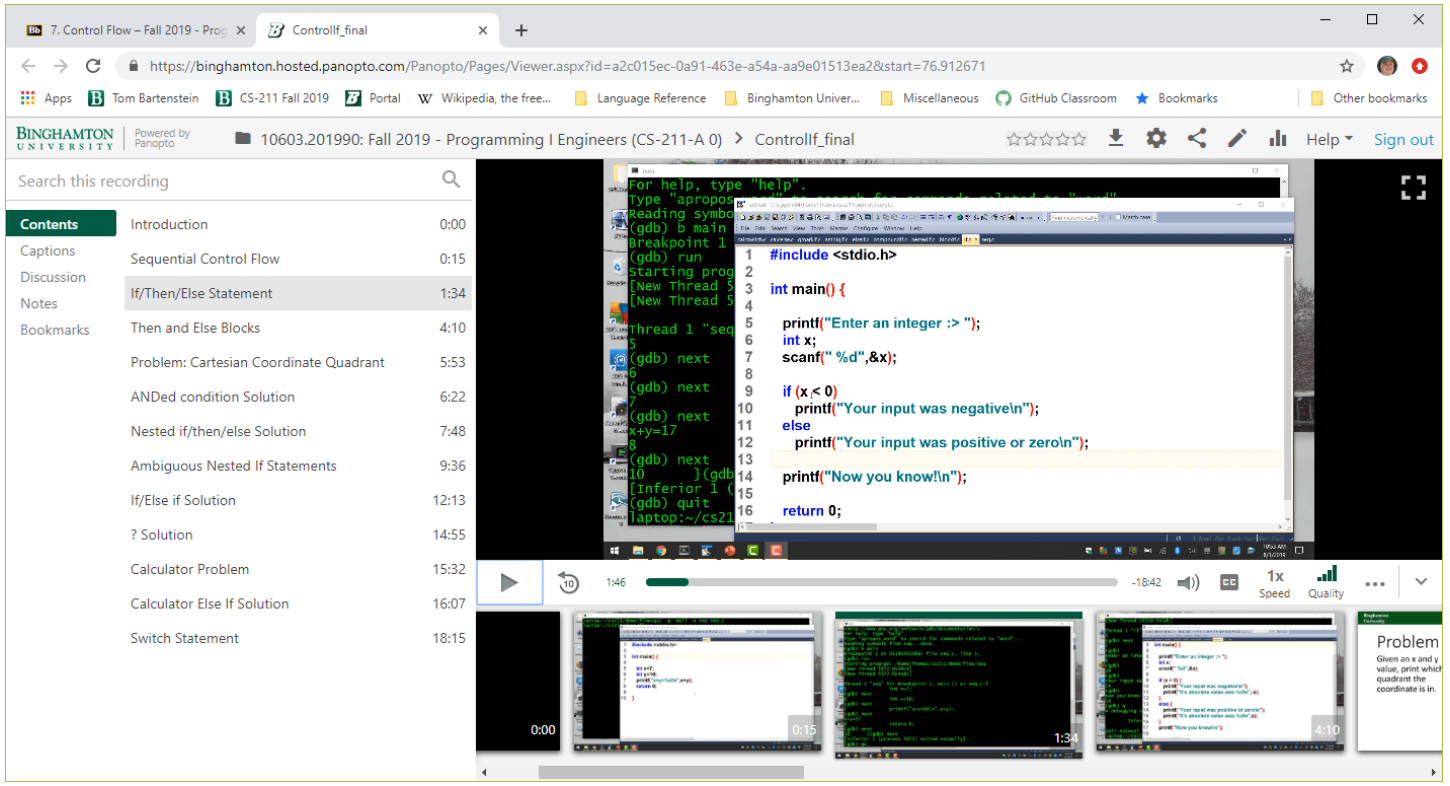
```
#include <stdlib.h>
#include <sys/times.h>
#include <stdio.h>
int main() {
    struct tms t; // Declare a time variable for the result
    srand(times(&t)); // Seed random with current time
    int n = rand() % 101;
    printf("Random number is %d\n",n);
    return 0;
}
```

Resources

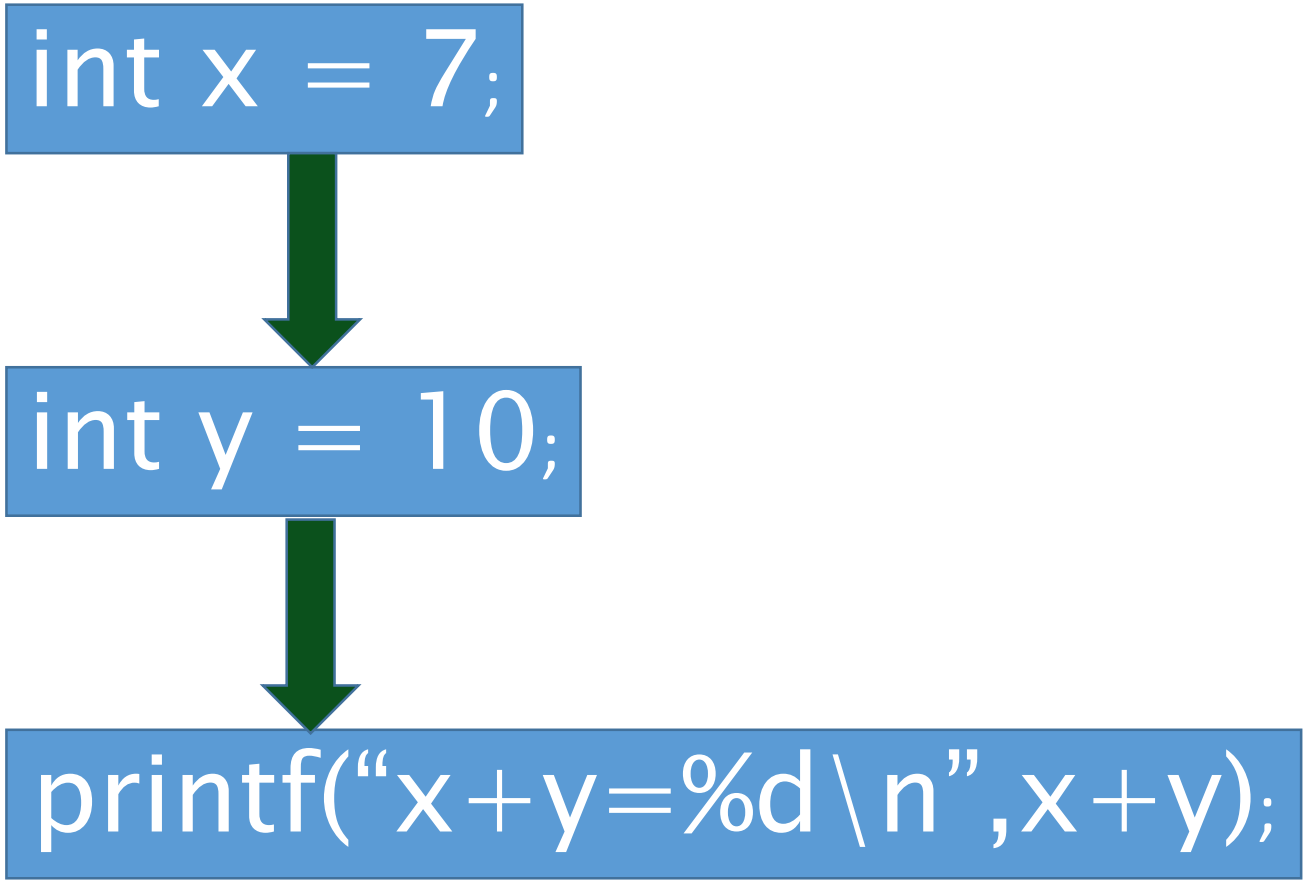
- Programming in C, Chapter 5
- Wikipedia: Conditional (computer programming) (https://en.wikipedia.org/wiki/Conditional_(computer_programming))
- Wikipedia: Control Flow (https://en.wikipedia.org/wiki/Control_flow)

Control Flow: If Statements

- Watch the video, available on myCourses
  - Content
    - Videos
      - 7. Control Flow



Sequential Control Flow



If

Conditional Processing

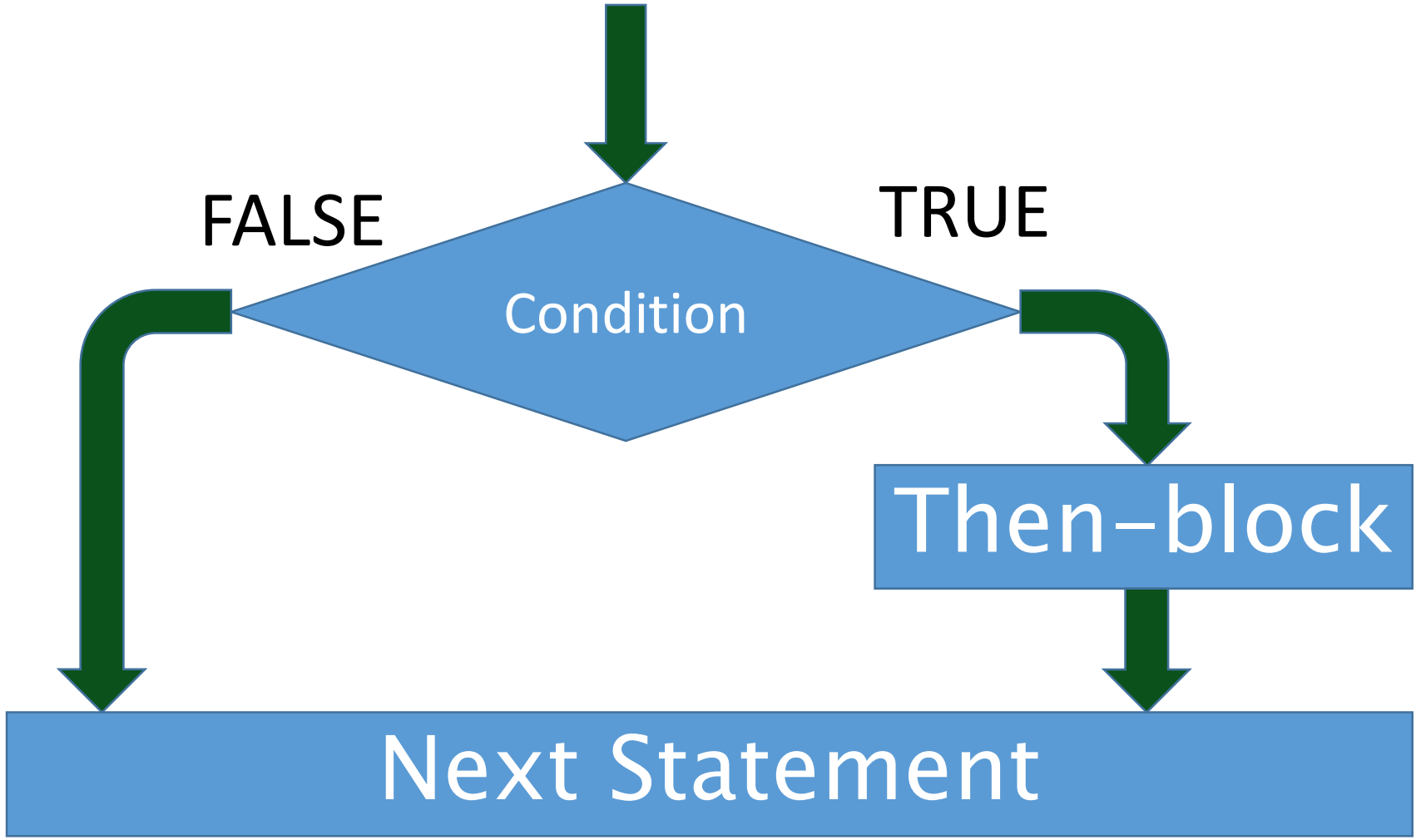
Simple If statement syntax

if (*condition*) *then_statement*;

- *condition* : Any expression whose results is true or false
- *then_statement* : Any statement or block of statements
- The *then_statement* is executed only when the <condition> is true
- Warning: No “then” keyword!

```
if (x!=0) x=113/x;
```

If statement flow



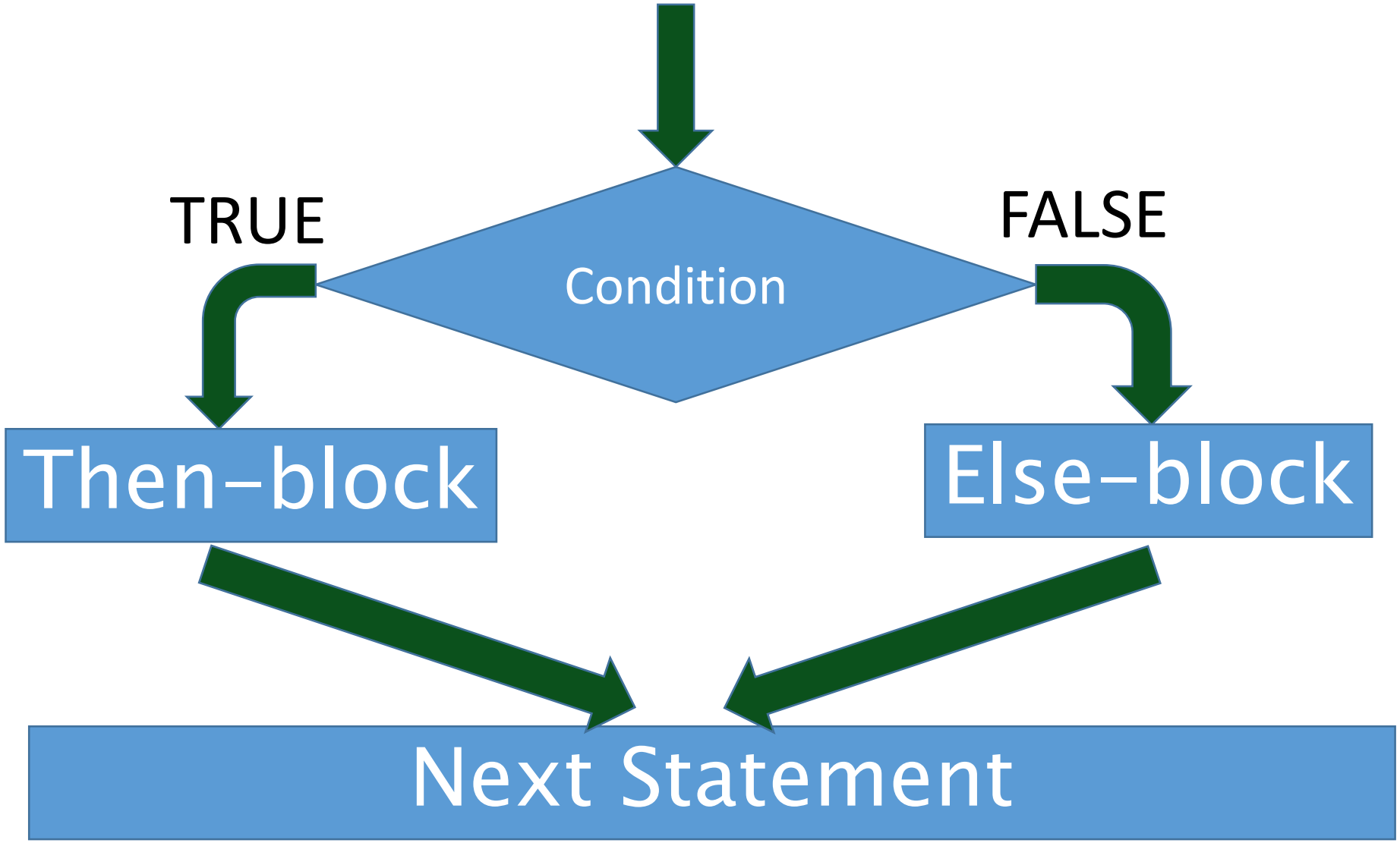
If/Then/Else syntax

if (*condition*) *then_statement* else *else_statement*

- *condition* : Any expression whose results is true or false
- *then_statement* : Any statement or block of statements
- *else_statement* : Any statement or block of statements
- The *then_statement* is executed only when the *condition* is true
- The *else_statement* is executed only when the *condition* is false

```
if (x!=0) x=113/x; else x=-1;
```

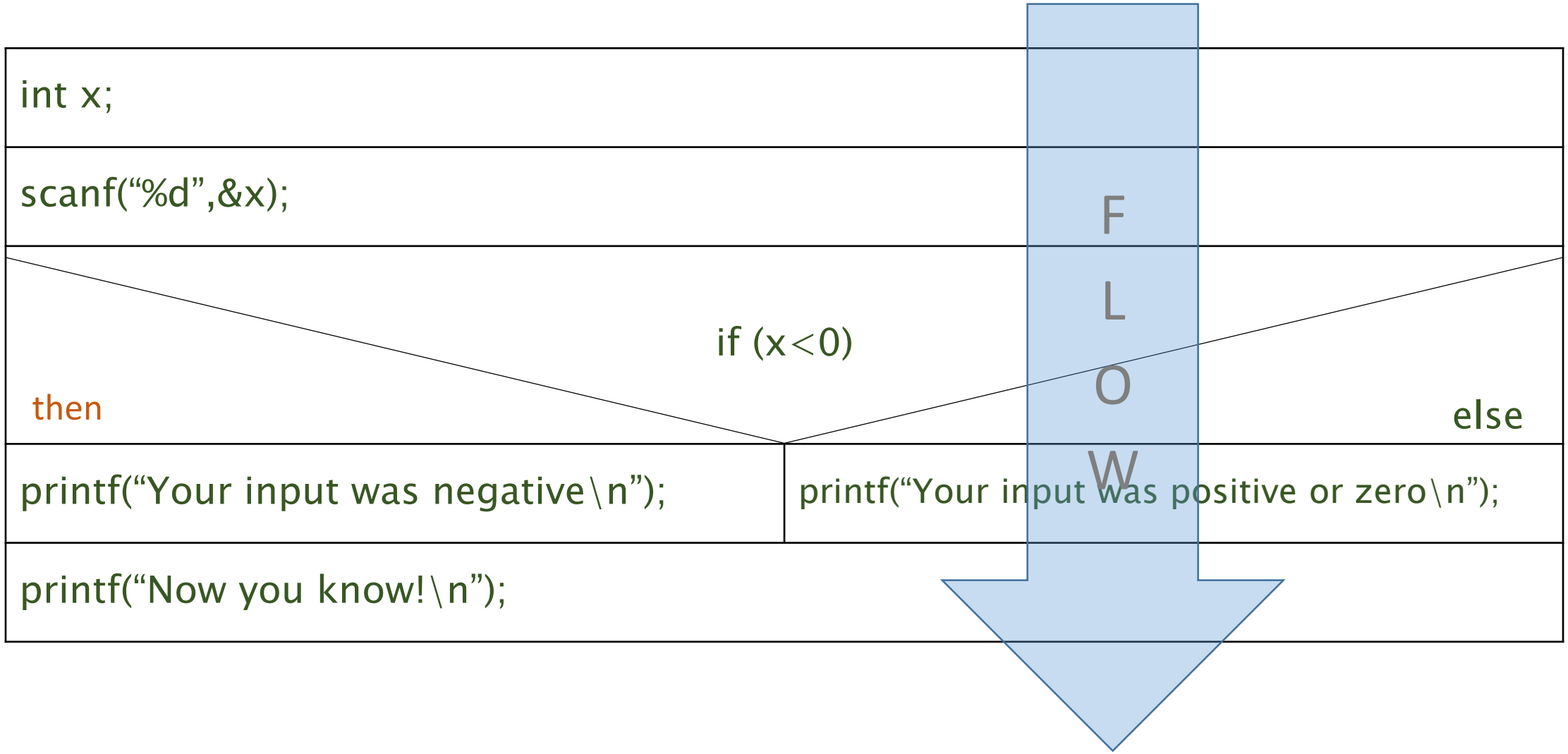
If / else statement flow



If/Then/Else Example

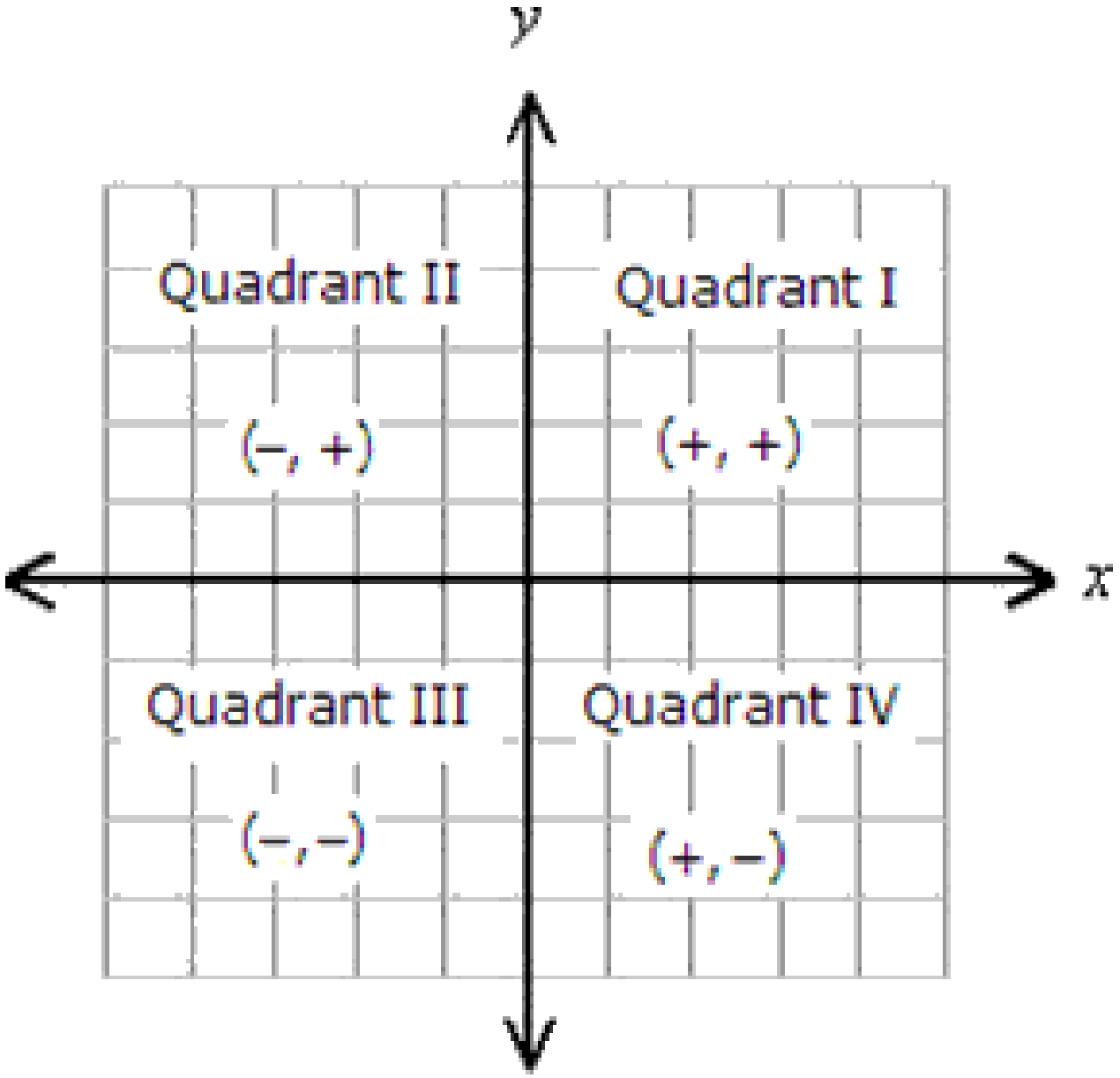
```
int x;
scanf("%d",&x);
if (x<0) {
        printf("Your input was negative\n");
} else {
        printf("Your input was positive or zero\n");
}
printf("Now you know!\n");
```

Nassi-Schneiderman…



Problem

Given an x and y value, print which quadrant the coordinate is in.



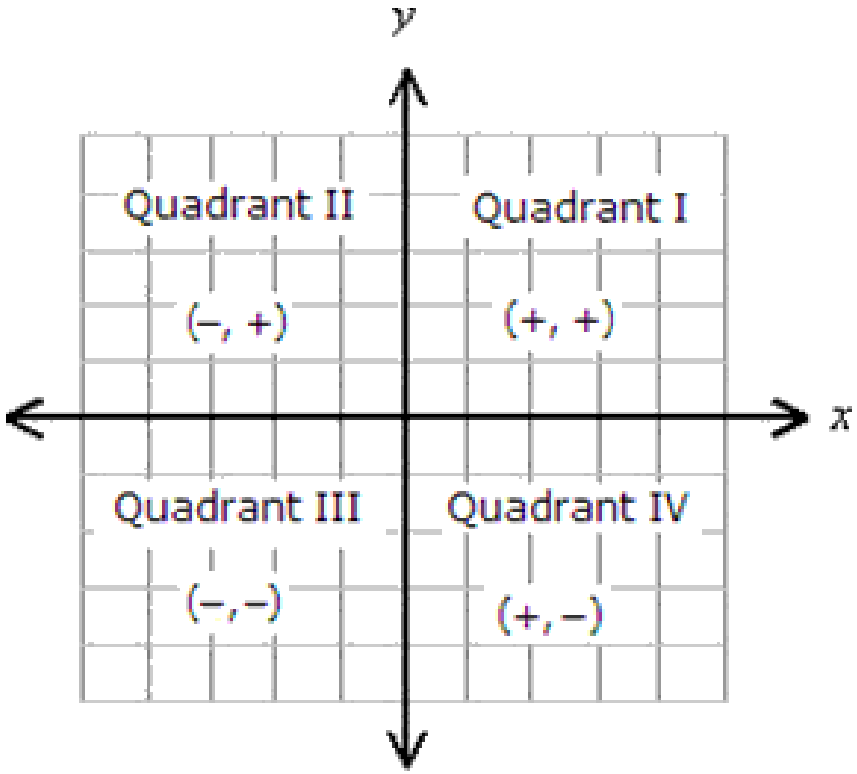
Compound Logic Solution

```
if ((x>=0)&&(y>=0)) printf (“Quadrant I”);
if ((x>=0)&&(y<0) ) printf(“Quadrant IV”);
if ((x<0) &&(y>=0)) printf(“Quadrant II”);
if ((x<=0)&&(y<0) ) printf(“Quadrant III”);
```

Nested If/Then/Else

- It's perfectly legal for a then block or else block to contain an if/then/else statement

```
if (x>=0) {
        if (y>=0) printf("Quadrant I");
        else printf("Quadrant IV");
} else {
        if (y>=0) printf("Quadrant II");
        else printf("Quadrant III");
}
```



Nested If/Then/Else Ambiguity

```
if (x>0) if (y>0) printf ("x>0, y>0\n");
else printf ("Is x negative, or is y negative????\n");

if (x>0) { if (y>0) printf ("x>0, y>0\n");
           else printf("y<0\n");
         }

if (x>0) { if (y>0) printf ("x>0, y>0\n"); }
else printf("x<0\n");
```

Without curly braces, "else" is matched with the nearest "if"

Alternative Else/If Construct

```
if      ((x>=0)&&(y>=0)) printf (“Quadrant I”);
else if ((x>=0)&&(y<0))    printf(“Quadrant IV”);
else if ((x<0)&&(y>=0))    printf(“Quadrant II”);
else                       printf(“Quadrant III”);
```

Deconstructing if/else/if

```
if ((x>=0)&&(y>=0)) { printf ("Quadrant I"); }
else {
        if ((x>=0)&&(y<0)) { printf("Quadrant IV"); }
        else {
                if ((x<0)&&(y>=0)) { printf("Quadrant II"); }
                else {
                        printf("Quadrant III");
                }
        }
}
}
```

Don't Forget "?"

```
if (doMore) {
        x=y*3;
} else {
        x=x-1;
}
```

when both *if_statement* and *else_statement* assign values to the same variable

```
x=doMore?(y*3):(x-1);
```

sometimes "?" helps show what is going on

Problem

- Write a C program to perform simple calculator functions
- Support +,-,*,/ operators and 2 arguments

Another else/if construct

comparing the same variable to different literal values

```
if      (op==‘+’) ans=a+b;
else if (op==‘-’)  ans=a-b;
else if (op==‘*’)  ans = a*b;
else if (op==‘/’)  ans = a/b;
else {
        printf “Unrecognized operator: %c\n”,op);
        ans=0;
}
```

Example Switch Statement

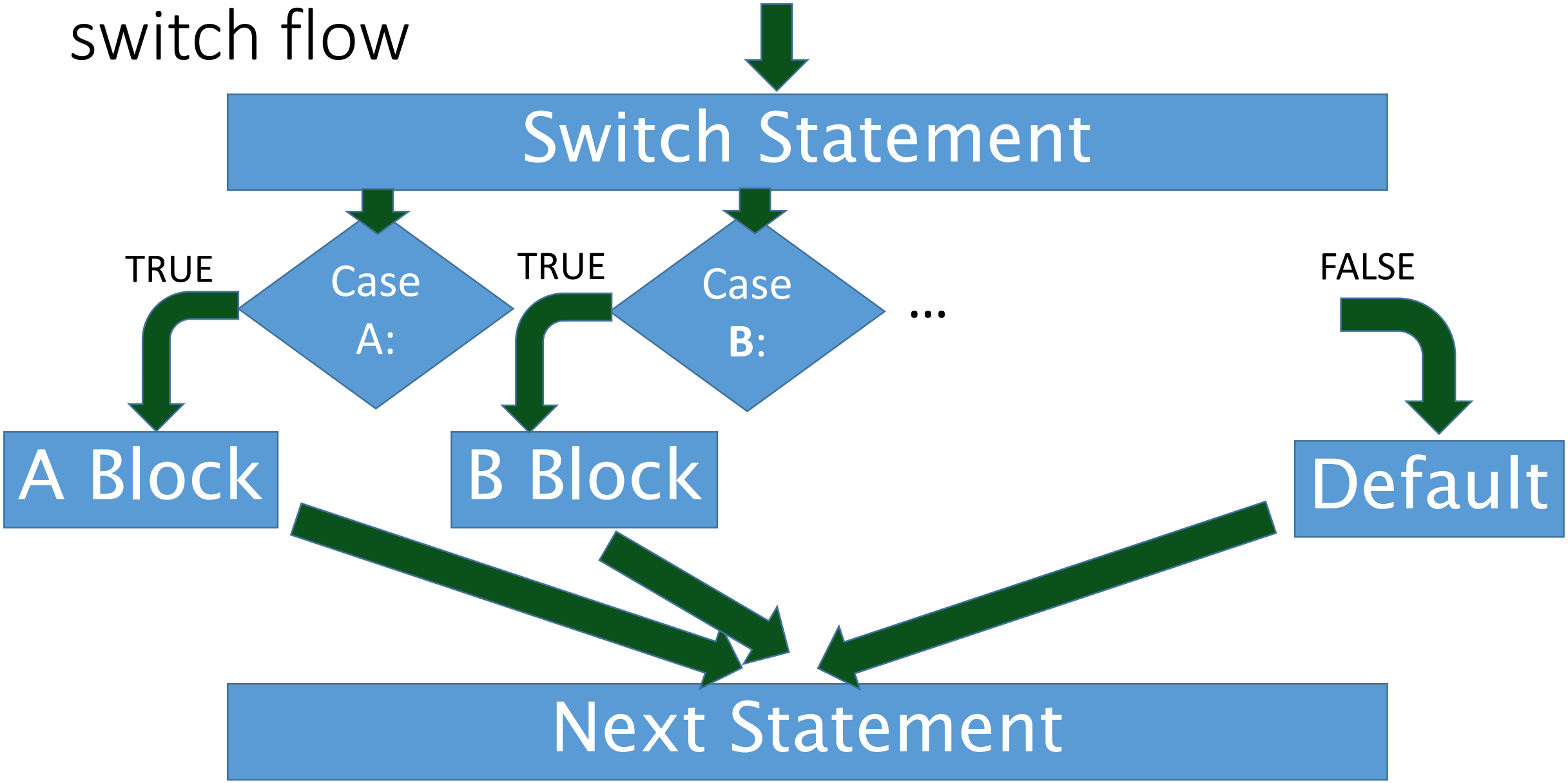

```
switch(op) {
        case('+'): ans=a+b; break;
        case('-'): ans=a-b; break;
        case('*'): ans=a*b; break;
        case('/'): ans=a/b; break;
        default: printf("Unrecognized operator: %c\n",op);
                ans=0;
}
```

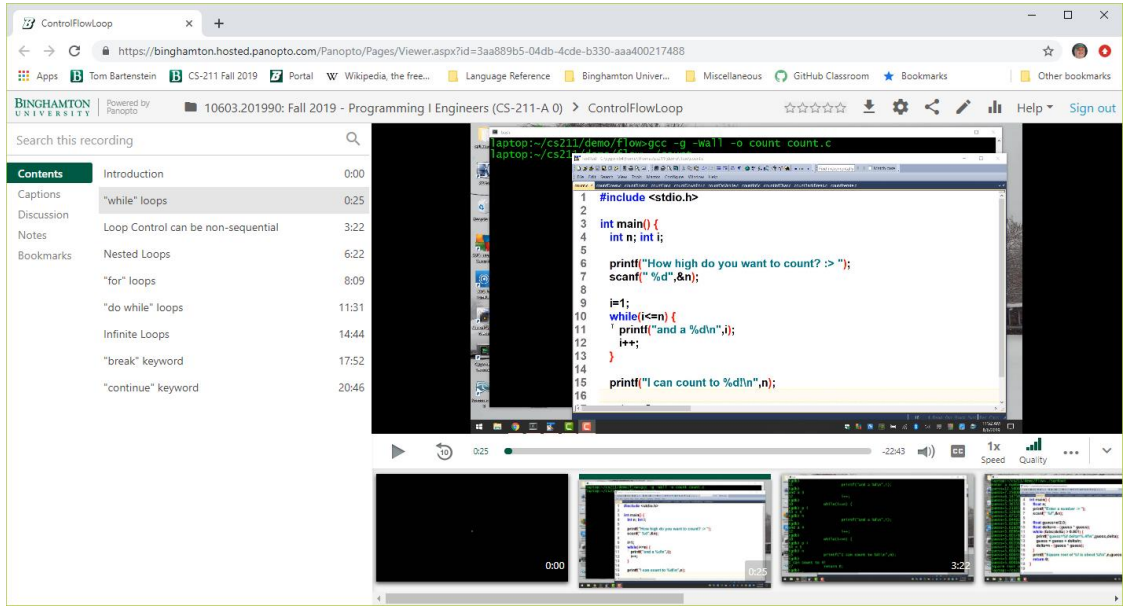
The "switch" statement

```
switch(expression) {
        case (v1) :
                v1_block
                break;
        case (v2) :
                v2_block
                break;
        …
        default:
                def_block
}
```

```
if (expression==v1) {
        v1_block
} else if (expression==v2) {
        v2_block
}
…
} else {
        def_block
}
```

switch flow


Switch Statement
TRUE
Case A:
TRUE
Case B:
...
FALSE
A Block
B Block
Default
Next Statement



Control Flow: Loops

Summary Notes

Loops

Going around in Circles

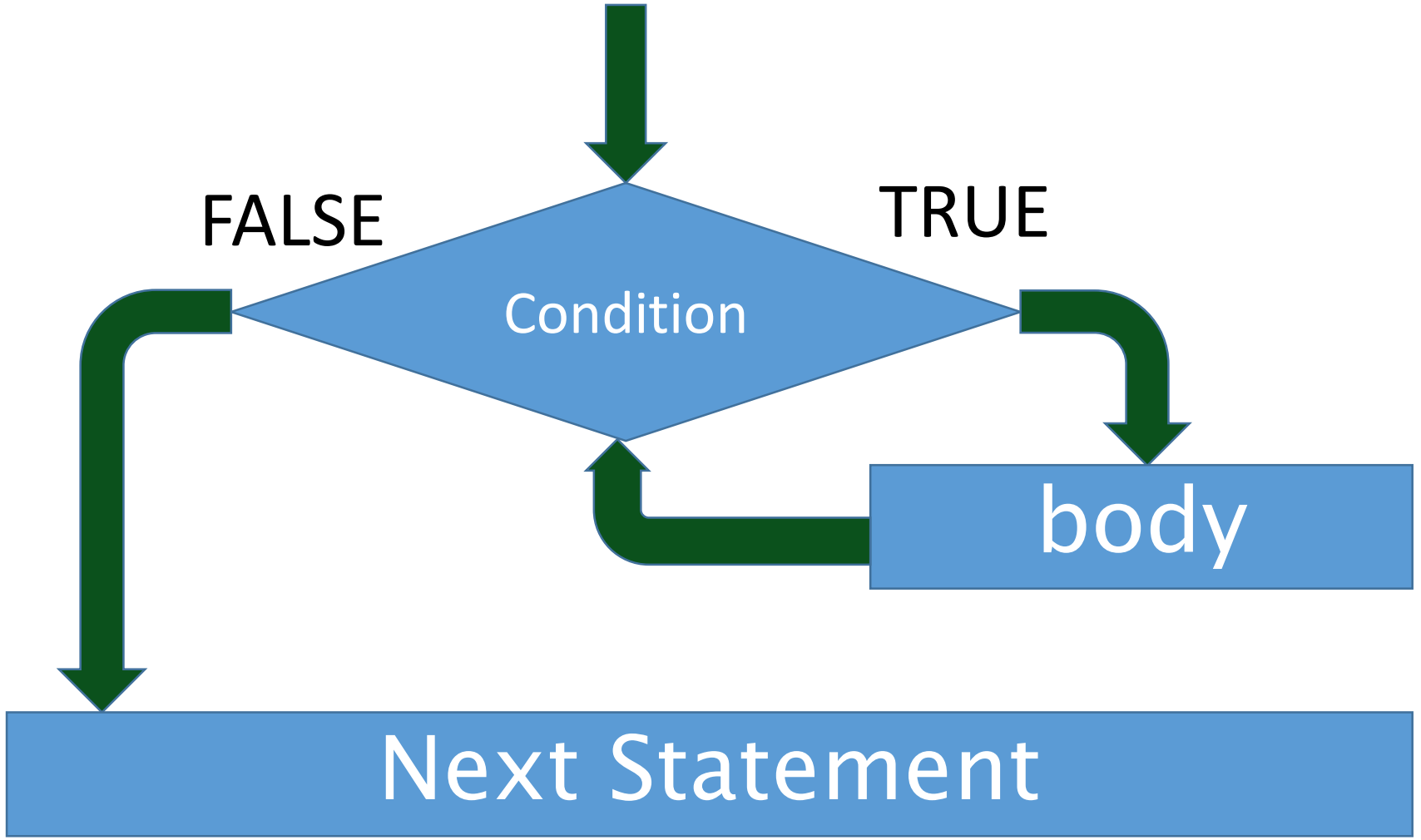
While Loop

while(*condition*) *body*

- *condition* : Any expression evaluated as true or false
- *body*: Any valid statement or block of statements

- *body* is re-executed as long as *condition* is true
- *condition* is re-evaluated after each "iteration" of the loop
- If *condition* is false to start with, *body* is never executed!

while statement flow



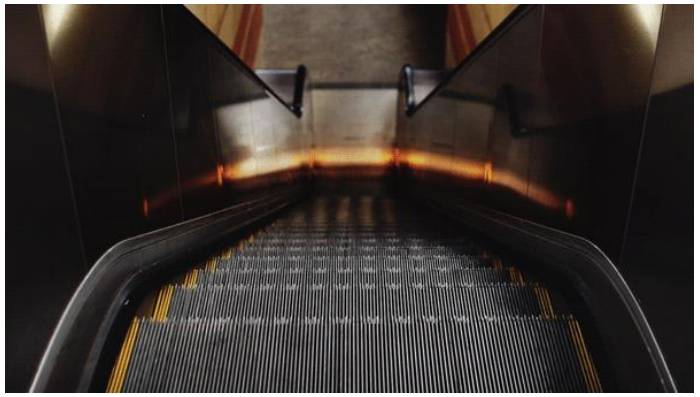
Example While Loop

```
int temp=check_temp();
while(temp<100) {
        add_heat();
        temp=check_temp();
}
/* temperature reached 100… water should boil */
```

Nested While Loops

```
int count=100;
while(count>0) {
        int result=experiment(count);
        printf("%3i : ",count);
        int j=0;
        while(j<result) {  printf("*"); j++; }
        printf("\n");
        count--;
}
```

```
100:
 99: *
 98: *
 97: **
 96: *
 95: **
 94: ***
 93: *****
 92: ******
 91: **********
 90: ***********
 89: ********
 88: ******
 87: ****
 86: ***
 85: **
 84: *
...
```



Example While Loop

```
int count=100;
while(count>0) {
        int result=experiment(count);
        printf("%3i : ",count);
        int j=0;
        while(j<result) {  printf("*"); j++; }
        printf("\n");
        count--;
}
```

Initialization

Condition

"Increment"

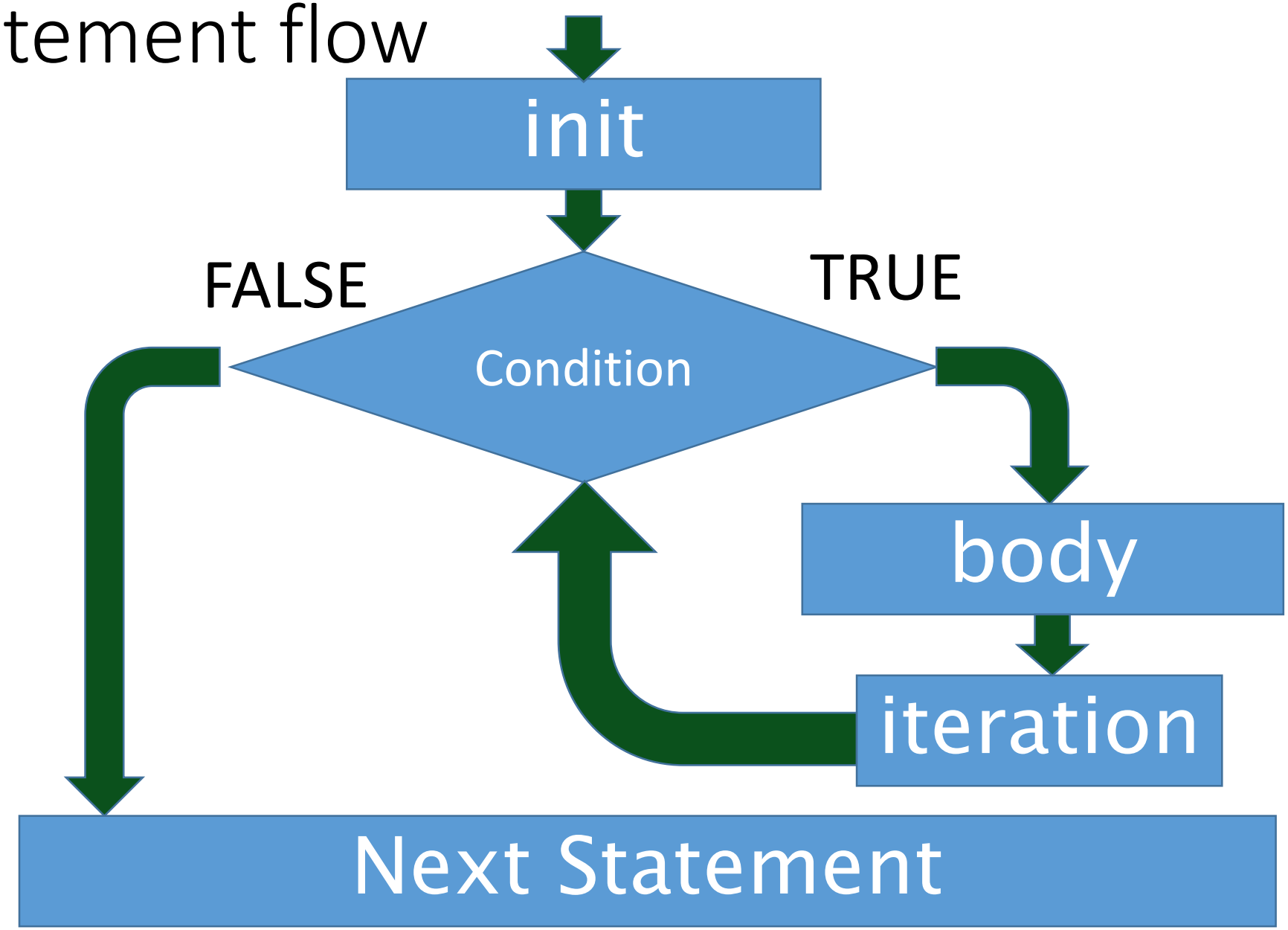
```
100:
 99: *
 98: *
 97: **
 96: *
 95: **
 94: ***
 93: *****
 92: ******
 91: **********
 90: ***********
 89: ********
 88: ******
 87: ****
 86: ***
 85: **
 84: *
...
```

for loop

for (*init*; *condition*; *iteration*) *body*

- *init* : Expression executed before loop starts
- *condition* : Expression evaluated to see if *body* should continue
- *iteration* : Expression executed after *body*, before *condition*
- *body* : Statement or block that makes up the body of the loop

for statement flow



Example for Loop

```
int count;
for(count=100;count>0;count--) {
        int result=experiment(count);
        printf("%3i : ",count);
        int j;
        for(j=0;j<result;j++) printf("*");
        printf("\n");
}
```

```
100:
 99: *
 98: *
 97: **
 96: *
 95: **
 94: ***
 93: *****
 92: ******
 91: **********
 90: ***********
 89: ********
 88: ******
 87: ****
 86: ***
 85: **
 84: *
...
```

More Example For Loops

```
for(data=get_first(); more_data(); data=get_next()) {
        process(data);
}

for(i=0,sum=0; i<100;) sum+=experiment(i++);
average=sum/100;
```

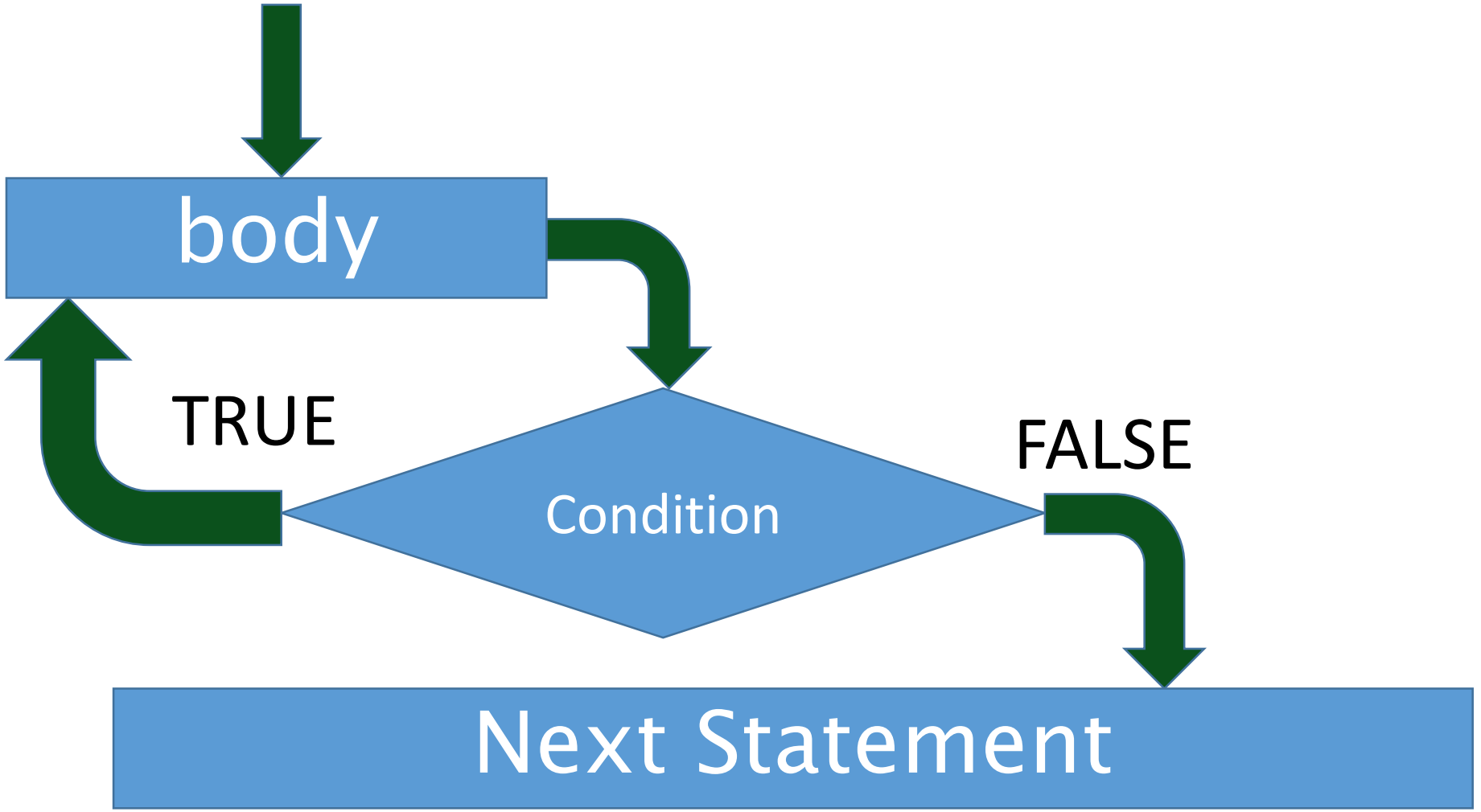
do/while loops

- C also supports a “do while” loop

```
do body while (condition) ;
```

- The only difference from a normal “while” loop is that the body of the loop is executed once BEFORE the condition is tested the first time.
- Very rarely used

do/while statement flow



Infinite Loops (or way too long)

```
int i=get_max();
while(i!=0) {
        process(i);
        i--;
}

for(i=0; i<10; i++) {
        process(i); i=3;
}

x=1;
while(x) { do_stuff(x); }
```

Breaking out of loops early

- break; statement leaves innermost loop (while/for/switch)
  - No checking of <condition>
  - No increment in for loop

```
for(i=0;i<100;i++) {
        result=experiment(i);
        if (result<0) break; /* Something bad happened */
        …
}
```

Early Iteration of Loops

- continue; statement ends *this* iteration of the loop
  - In for loops, iteration statement executed again
  - condition re-evaluated
  - If condition is true, next iteration of the loop starts

```
for(count=0; count<100; count++) {
        if (0==count%7) continue; // skip every 7th experiment
        result=experiment(count);
        …
}
```